Leaking addresses with vulnerabilities that can’t read good

pakt\textsuperscript{1} Dionysus Blazakis\textsuperscript{2}

\textsuperscript{1}Department of efficient pie consumption
Sourcefire

\textsuperscript{2}Trapbit

June 5, 2013
Outline

1. Intro
   - What are info leaks?
   - Timing attacks
   - Previous Work
   - What we did

2. Timing attack on hash tables in Firefox
   - Setting the stage
   - The trick
   - Demo

3. Conservative GC for memory disclosure
Introduction

What are info leaks?
- Timing attacks
- Previous Work
- What we did

Timing attack on hash tables in Firefox
- Setting the stage
- The trick
- Demo

Conservative GC for memory disclosure

pakt, Dion Blazakis
Leaking addresses with vulnerabilities that can’t read good
Possible definitions

- Any procedure that reveals exact values of target’s memory?
- How about learning a certain value (like a crypto key stored in mem) must be odd?
- That’s a leak – we halved the search space by 2!
-Leaks need a more abstract definition...
Entropy to the rescue

- Informally, entropy measures how much information is necessary to describe something
- Eg: \([a, b]\) with \(a, b \in \{0, 1\}\) requires 2 bits to completely describe
- Assuming \(a \oplus b = 1\), we need only 1 bit – given \(a\), \(b = a \oplus 1\)
- Additional assumption reduced number of all possible states from 4 to 2
- In other words, additional information lowered the system’s entropy
We leak information when...

- we reduce the entropy of target’s state (memory, registers, ...)
- it doesn’t matter how we do it
Outline

1. Intro
   - What are info leaks?
   - Timing attacks
   - Previous Work
   - What we did

2. Timing attack on hash tables in Firefox
   - Setting the stage
   - The trick
   - Demo

3. Conservative GC for memory disclosure

Leaking addresses with vulnerabilities that can’t read good
Definition (Timing attack)

In cryptography, a timing attack is a side channel attack in which the attacker attempts to compromise a cryptosystem by analyzing the time taken to execute cryptographic algorithms.

- Simplest example: `strcmp(input, "SECRET")`
Outline

1. **Intro**
   - What are info leaks?
   - Timing attacks
   - Previous Work
   - What we did

2. **Timing attack on hash tables in Firefox**
   - Setting the stage
   - The trick
   - Demo

3. **Conservative GC for memory disclosure**

---

Leaking addresses with vulnerabilities that can’t read good
Previous work

- Plenty in crypto and it sec (enumerating users, learning crypto keys, blind SQLi, ...)
- None in ASLR context
- History:
  - Leaking information using timing attacks on hash tables (pakt) [1]
  - Practical timing side channel attacks against kernel space ASLR (Hund, Willems, Holz) [2]
  - Conservative GC for Memory Disclosure (Dion) [3]
  - Epic "cnot" Writeup (Ron Bowes) – timing attack against a crackme [4]
Outline

1. Intro
   - What are info leaks?
   - Timing attacks
   - Previous Work
   - What we did

2. Timing attack on hash tables in Firefox
   - Setting the stage
   - The trick
   - Demo

3. Conservative GC for memory disclosure
What we did

- Timing attack on hash table implementation in Firefox (pakt)
  - leaks a pointer to a string object
  - still unpatched :)
- Conservative GC manipulation (Dion)
  - uses the garbage collector to leak pointers to various objects
Outline

1. Intro
   - What are info leaks?
   - Timing attacks
   - Previous Work
   - What we did

2. Timing attack on hash tables in Firefox
   - Setting the stage
   - The trick
   - Demo

3. Conservative GC for memory disclosure

pakt, Dion Blazakis
Leaking addresses with vulnerabilities that can’t read good
Setting the stage

Disclaimer

Some details were omitted or simplified for brevity. Proofs aren’t provided for the same reason.

- We’re after hash tables holding properties of JS objects
- Executing `obj[x]=0;` results in $T[h(x)] = 0$, where $T$ is the object’s hash table
- `obj.hasOwnProperty(x)` allows us to check if $T[h(x)]$ is set, without walking up the inheritance chain
- The problem (for Firefox) is that $x$ can be either a pointer, or a user supplied integer (oops)
Double hashing

\[ h(k, j) = h_1(k) + jh_2(k) \mod |T| \]
Definitions

- $T$ – hash table that all of the below functions work on
- $LT(k)$ – lookup time of $k$, measured with JS clock (Date object), implemented as `obj.hasOwnProperty(k)`
- $CL(k)$ – length of $k$’s chain (number of collisions)
Outline

1. **Intro**
   - What are info leaks?
   - Timing attacks
   - Previous Work
   - What we did

2. **Timing attack on hash tables in Firefox**
   - Setting the stage
   - The trick
   - Demo

3. **Conservative GC for memory disclosure**
Integers in the hash table

- The hash function is easy to invert (multiplication mod $2^{32}$)
- Use $h^{-1}$ to populate $T$ like in the image above
- We know which integer is stored in what slot
- We need to leave some slots free, to prevent Firefox from reallocating $T$
Keep querying $T$ with different pointers
(obj.hasOwnProperty(x))

At some point, we’ll find $x$, for which $LT(x)$ is noticeably longer than for others
Let’s call this object $M_{str}$ and let $M_t = LT(M_{str})$

Recall, $h(k, j) = h_1(k) + jh_2(k)$

Firefox splits $k$ into two keys: $k_1, k_2$:
$h(k, j) = h_1(k_1) + jh_2(k_2)$ (imagine $k_1$ is the low word, $k_2$ high word of $k$)

We need to learn both $h_1(k_1)$ and $h_2(k_2)$, invert them and only then we can recover $k$ ($M_{str}$’s pointer)
Finding $h_2(k_2)$

Let $r = h_2(k_2)$

Notice that $\forall i, j < CL(k) \ r | h(k, i) - h(k, j)$ (2nd case explained in the blog post)
Finding $h_2(k_2)$ II

- Indeed: $h(k, i) - h(k, j) = h_1(k) - ih_2(k) - h_1(k) + jh_2(k) = h_2(k)(j - i) = r(j - i)$
- If we collect enough of chain’s elements, we can compute their differences and take $r = h_2(k_2) = \gcd(diffs)$ (gcd - greatest common divisor)
- But we won’t always get the correct result...
Counterexample

- Let \( h_2(k) = 1 \)
- Taken slots: \( \{1, 2, 3, 4, 5, 6\} \)
- Random subset: \( R = \{2, 4, 6\}, \ D(R) = \{2, 4\} \)
- \( \gcd(D(R)) \neq h_2(k)! \)
The problem

- Let the chain of $M_{str} = \{a_0, a_0 + r, \ldots, a_0 + (n - 1)r\}$
- Then the set of differences is $D = \{(i - j)r : i, j < n\}$
- $\gcd(D) = r \times \gcd(\{(i - j) : i, j < n\})$
- We might get something larger than $r$...
What are the odds?

Theorem (Nymann, 1972)

$k$ integers chosen independently and uniformly from \{1, ..., $n$\}, are coprime with probability $1/\zeta(k)$.

$$\zeta(k) = 1 + \frac{1}{2^k} + \frac{1}{3^k} + \frac{1}{4^k} + \frac{1}{5^k} + \ldots$$
What does this mean for us?

\[ \zeta(k) = 1 + \frac{1}{2^k} + \frac{1}{3^k} + \frac{1}{4^k} + \frac{1}{5^k} + \ldots \]

\[ \lim_{k \to +\infty} 2^k (\zeta(k) - 1) = 1 \]

\[ \zeta(k) - 1 \approx \frac{1}{2^k} \]

\[ \zeta(k) \approx 1 + \frac{1}{2^k} \]

\[ \frac{1}{\zeta(k)} \approx \frac{1}{1 + \frac{1}{2^k}} \]

As \( k \) increases, \( \frac{1}{\zeta(k)} \) rapidly approaches 1
Finding chain elements I

- Remove an integer from $T$ and measure $LT(M_{str})$
- If we hit a slot that belongs to $M_{str}$’s chain, $LT(M_{str})$ will drop
Finding chain elements II

- We can’t distinguish slots that don’t belong to the chain from elements at the very end of it (removing last element is hard to measure)
- Solution: restrict ourselves to the first half – accept only integers that cause $LT(M_{str}) < M_t/2$
Removing elements one by one is too slow

Let’s remove $k$ random elements

Probability that we failed to hit any element of $M_{str}$’s chain is $\left(1 - \frac{CL(M_{str})}{|T|}\right)^k$
Exponent works in our favor, but how to estimate an optimal $k$?

Failing with $(1/2)^{30}$ probability is as good as $(1/2)^{100}$ failure, but the second one is slower (requires more removals)
We know $T$’s layout, since we filled it ourselves.

We can compute all chains in $T$ in javascript and measure how long it takes to traverse them.

Using these measurements as data points, we can build a linear regression model.

This will provide a function $g(x) = ax + b$ that maps chain’s length to its lookup time.

By inverting that function, we can map lookup times to chain lengths.
Linear regression model

Chain length vs. lookup time

Observed vs. Predicted
Optimal number of elements to remove

- \( g^{-1}(LT(M_{str})) = CL(M_{str}) \)
- Having \( CL(M_{str}) \) we can pick \( k \), so that
  \( (1 - CL(M_{str})/|T|)^k < \epsilon \), for any \( \epsilon > 0 \)
- If, after removing \( k \) random keys, \( LT(M_{str}) \) drops below \( M_t/2 \), we know we hit something on the chain
- We need to know which of the \( k \) keys belongs to the chain
- Just bisect in \( O(logk) \)
Finding the starting point

Binary search

- Remove $k_0$, $k_0 + r$ and measure $LT(M_{str})$
- There are three cases to consider:
  - removing 5 or 7 won’t affect $LT(M_{str})$ (outside)
  - removing 4 or 6 will cause $LT(M_{str}) < M_t/2$ (inside)
  - removing 8 will cause $LT(M_{str}) < M_t/2$ and removing 10 will have no effect on $LT(M_{str})$ (hit)
- Use these three cases to bisect
We found both $r = \gcd(\text{diffs}) = h_2(k_2)$ and $h_1(k_1)$.

Inverting these functions is easy.

Recover $k_1$, $k_2$ and compute $k = \text{pointer}(M_{str})$ :)
Outline

1 Intro
   - What are info leaks?
   - Timing attacks
   - Previous Work
   - What we did

2 Timing attack on hash tables in Firefox
   - Setting the stage
   - The trick
   - Demo

3 Conservative GC for memory disclosure

Leaking addresses with vulnerabilities that can’t read good
Demo

- Worked on FF 14, works on FF 21
- Low risk, unpatched for 10 months
- https://github.com/pakt/exp-dev/
Timing attack on hash tables in Firefox

Conservative GC for memory disclosure

Leaking addresses with vulnerabilities that can’t read good
For Further Reading

- **pakt**
  Leaking information with timing attacks on hashtables
  2012

- **Ralf Hund, Carsten Willems, Thorsten Holz**
  Practical Timing Side Channel Attacks Against Kernel Space ASLR
  [http://www.reddit.com/r/netsec/comments/1a2kv0/practical_timing_side_channel_attacks_against/](http://www.reddit.com/r/netsec/comments/1a2kv0/practical_timing_side_channel_attacks_against/)

- **Dionysus Blazakis**
  Conservative GC for Memory Disclosure
  [https://github.com/justdionysus/gcwoah](https://github.com/justdionysus/gcwoah)

- **pakt, Dion Blazakis**
  Leaking addresses with vulnerabilities that can’t read good
Ron Bowes
Epic "cnot" writeup
http://www.skullsecurity.org/blog/2013/epic-cnot-writeup-plaidctf